Security: CDSA and SecureTransport

Session 113

Security: CDSA and SecureTransport

Craig Keithley
Security and Cryptography Technology Evangelist

Introduction

- Using Common Data Security Architecture
- Using CDSA for cryptography
- Using SecureTransport

Security: CDSA and SecureTransport

John Hurley, Ph.D.
Security Policy Architect

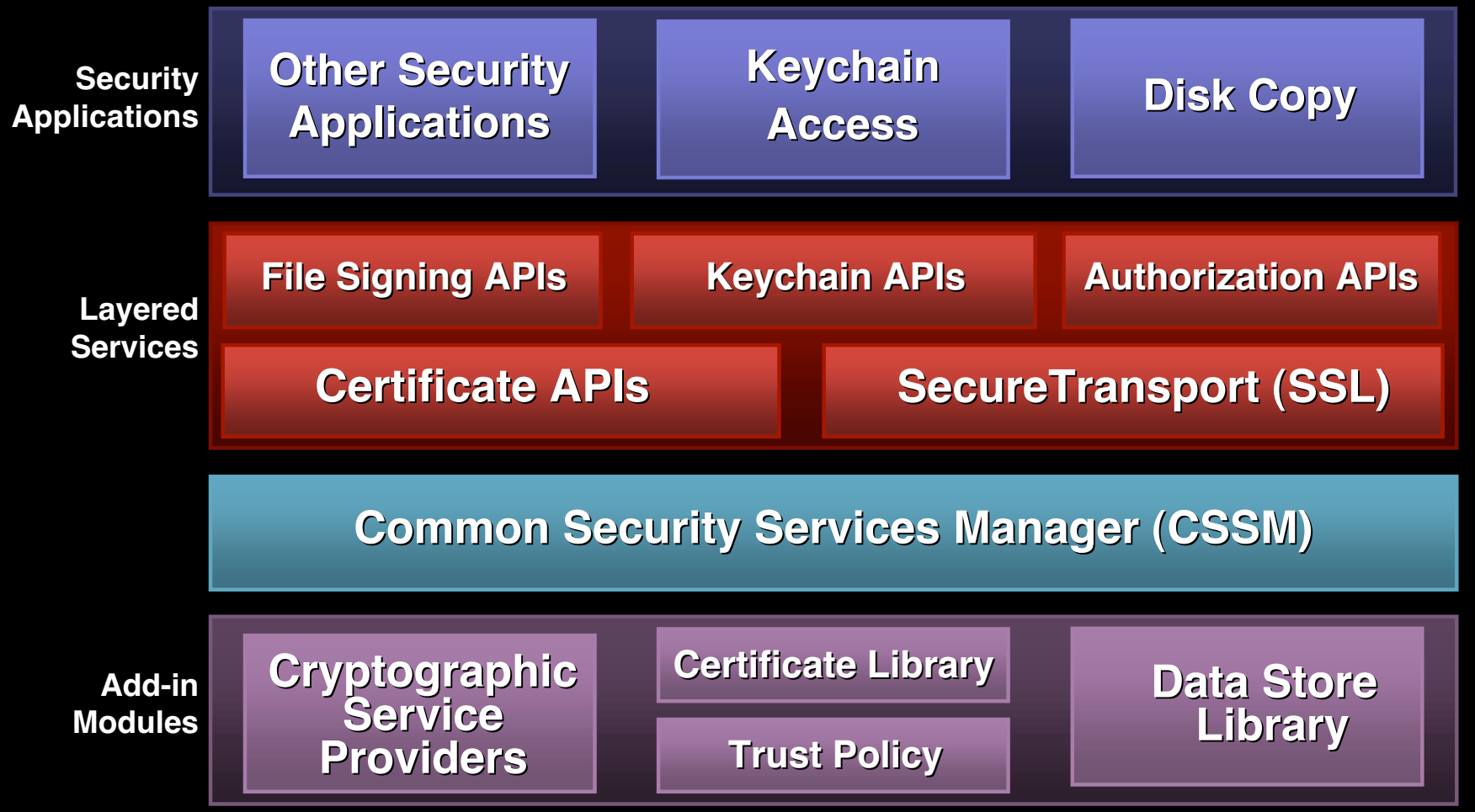
What You Will Learn

- Overview of the CDSA architecture
- Using CDSA directly for cryptography
- Using CDSA for SSL

CDSA

- Common Data Security Architecture
- OpenGroup standard
- Foundation for cryptography and Public Key Infrastructure

CDSA Architecture



Data Library Modules (DLs)

- Store information used by other CDSA modules
- Apple File CSP/DL
  - A combination CSP/DL (encrypts and stores)
  - Keychains are flat files maintained by this CSP/DL
- Other examples
  - LDAP DL

Certificate Library Modules (CLs)

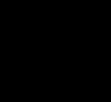
- Interpret public key certificates
- Examples
    - Apple CL interprets X.509 v3 certificates
    - PGP CL
    - Attribute CL

Trust Policy Modules (TPs)

- Encapsulate how certificates should be evaluated for trust decisions
- Examples
    - A corporate TP could reject any certificate chain that did not contain the corporate certificate in the chain
    - Could implement web-of-trust instead of chain of trust

CDSA vs. OpenSSL

- Apple encouraging use of CDSA over OpenSSL
- Advantages of CDSA
  - More secure
  - Easier to use
  - Better integration with other applications and services
- Making more sample code available



Security: CDSA and SecureTransport

Richard Murphy
Manager, Data Security

Using CDSA for Encryption

- Initialize some local variables
- Create a CSSM_KEY
- Copy the caller's key material
- Delete the key generation context
- Set up an encryption context
- Do the encryption
- Deal with possible remaining data
- Delete the encryption context

Using CDSA for Encryption

- Function header
- Here we assume key bits passed in

```
CSSM_RETURN doEncrypt(
    CSSM_CSP_HANDLE             cspHandle,
    const  CSSM_DATA            *plainText,
    CSSM_DATA_PTR               cipherText,
    unsigned char               *keyData,
    unsigned                    keyDataLen)
{
```

Using CDSA for Encryption

- Initialize some local variables

```
CSSM_KEY                aesKey;
CSSM_CC_HANDLE          ccHandle;
uint32                  keyLen;
CSSM_DATA label = {6, (uint8 *)"sample"};
CSSM_DATA iv = {16, (uint8 *)"some init vector"};
uint32                  bytesEncrypted;
CSSM_DATA remData = {0, NULL};
```

Using CDSA for Encryption

- Create a CSSM_KEY

```
/* first create a CSSM_KEY */
CSSM_CSP_CreateKeyGenContext(cspHandle,
        CSSM_ALGID_AES,
        128,                    // key size in bits
        NULL,                   // Seed
        NULL,                   // Salt
        NULL,                   // StartDate
        NULL,                   // EndDate
        NULL,                   // Params
        &ccHandle);
```

Using CDSA for Encryption

- Copy the caller's key material

```
/* copy caller's key material */
      len =  keyDataLen;
      if (len > aesKey.KeyData.Length) {
            len = aesKey.KeyData.Length
}
memmove(aesKey.KeyData.Data, keyData, len);
aesKey.KeyData.Length = len;

/* delete the key generation context */
CSSM_DeleteContext(ccHandle);
```

Using CDSA for Encryption

- Generate the key

```
CSSM_GenerateKey(ccHandle,
        CSSM_KEYUSE_ANY,
        CSSM_KEYATTR_RETURN_DATA |
        CSSM_KEYATTR_EXTRACTABLE,
        &label,
        NULL,                   // ACL
        &aesKey);
```

Using CDSA for Encryption

- Set up an encryption context

```
/* set up an encryption context */
    CSSM_CSP_CreateSymmetricContext(cspHandle,
        CSSM_ALGID_AES,
        CSSM_ALGMODE_CBCPadIV8,
        NULL,                   // for ACL
        &aesKey,
        &iv,                    // initialization vector
        CSSM_PADDING_PKCS7,
        NULL,
        &ccHandle);
```

Using CDSA for Encryption

- Do the encryption

```
/* perform encryption - CSP mallocs the ciphertext */
cipherText->Data = NULL;
cipherText->Length = 0;
CSSM_EncryptData(ccHandle,
        plainText,
        1,
        cipherText,
        1,
        &bytesEncrypted,
        &remData);
    outData.Length = bytesEncrypted;
```

Using CDSA for Encryption

- Process any remaining data and clean up

```
if (remData.Length != 0) {
    /* append remaining data to outData */
    uint32 newLen = remData.Length + outData.Length;
    outData.Data = (uint32 *)realloc(outData.Data, newLen);
    memmove (outData.Data + outData.Length,
                remData.Data,
                remData.Length);
    free(remData.Data);
}

CSSM_DeleteContext(ccHandle);
return CSSM_OK;
```

Using SecureTransport

- Main Features
  - SSLv2, SSLv3, TLSv1 protocol support
  - Client- and Server-side support using identical API
  - Application performs no cryptographic operations and needs no knowledge of SSL/TLS protocol
  - Transport layer independence—works with UNIX sockets, OpenTransport, etc., in both synchronous and nonblocking modes
  - Straightforward, traditional Mac-style ANSI "C" API

Using SecureTransport

- Benefits
  - Certificate verification and trust evaluation performed by common Mac OS X Security code, allowing user intervention and evaluation when untrusted server certificates are found
  - Server-side certificate and key handling is performed using Keychain and SecureServer; private key material is never exposed to the application

Benefits

- Allows, but does not require, application code to specify all negotiated protocol parameters, and/or to examine those parameters upon completion of negotiation
- Comprehensive set of CipherSuite implementations ensuring interoperability with a wide variety of existing SSL clients and servers

Using SecureTransport

- APIs
  - CFNetwork
  - URL Access
- Other applications
  - Mail

SecureTransport Functions

- Setup

```
OSStatus SSLNewContext (Boolean  isServer, SSLContextRef *contextPtr);
OSStatus SSLDisposeContext (SSLContextRef context);
OSStatus SSLGetSessionState (SSLContextRef context, SSLSessionState *state);
OSStatus SSLSetIOFuncs (SSLContextRef context, SSLReadFunc  read, SSLWriteFunc write);
OSStatus SSLSetProtocolVersion (SSLContextRef  context, SSLProtocol version);
OSStatus SSLGetProtocolVersion (SSLContextRef context, SSLProtocol *protocol);
OSStatus SSLSetCertificate (SSLContextRef context, CFArrayRef certRefs);
OSStatus SSLSetConnection (SSLContextRef context, SSLConnectionRef connection);
```

SecureTransport Functions

- Attributes

```
OSStatus SSLSetPeerDomainName (SSLContextRef context, const char *peerName, size_t peerNameLen);
OSStatus SSLGetPeerDomainNameLength (SSLContextRef context, size_t *peerNameLen);
OSStatus SSLGetPeerDomainName (SSLContextRef context, char *peerName, size_t *peerNameLen);
OSStatus SSLGetNegotiatedProtocolVersion (SSLContextRef context, SSLProtocol *protocol);
OSStatus SSLGetNumberSupportedCiphers (SSLContextRef context, size_t *numCiphers);
OSStatus SSLGetSupportedCiphers  (SSLContextRef context, SSLCipherSuite *ciphers, size_t *numCiphers);
OSStatus SSLSetEnabledCiphers (SSLContextRef context, const SSLCipherSuite *ciphers, size_t numCiphers);
OSStatus SSLGetNumberEnabledCiphers  (SSLContextRef context, size_t *numCiphers);
OSStatus SSLGetEnabledCiphers (SSLContextRef context, SSLCipherSuite *ciphers, size_t *numCiphers);
```

SecureTransport Functions

- Options

```
OSStatus SSLSetAllowsExpiredCerts (SSLContextRef context, Boolean allowsExpired);
OSStatus SSLGetAllowsExpiredCerts (SSLContextRef context, Boolean *allowsExpired);
OSStatus SSLSetAllowsAnyRoot (SSLContextRef context, Boolean anyRoot);
OSStatus SSLGetAllowsAnyRoot (SSLContextRef context, Boolean *anyRoot);
```

SecureTransport Functions

- Protocol

```
OSStatus SSLGetPeerCertificates (SSLContextRef  context, CFArrayRef *certs);
OSStatus SSLSetPeerID (SSLContextRef  context, const void  *peerID, size_t peerIDLen);
OSStatus SSLGetPeerID (SSLContextRef  context, const void  **peerID, size_t *peerIDLen);
OSStatus SSLGetNegotiatedCipher (SSLContextRef  context, SSLCipherSuite  *cipherSuite);
OSStatus SSLSetEncryptionCertificate (SSLContextRef context, CFArrayRef certRefs);
OSStatus SSLSetClientSideAuthenticate  (SSLContextRef context, SSLAuthenticate auth);
OSStatus SSLHandshake (SSLContextRef context);
```

SecureTransport Functions

- I/O

```
OSStatus SSLWrite (SSLContextRef context, const void * data, size_t dataLength, size_t  *processed);
OSStatus SSLRead (SSLContextRef context, void * data, /* RETURNED */ size_t dataLength, size_t  *processed);
OSStatus SSLGetBufferedReadSize (SSLContextRef context, size_t *bufSize);
OSStatus SSLClose (SSLContextRef context);
typedef OSStatus (*SSLReadFunc)  (SSLConnectionRef  connection, void  *data,  size_t  *dataLength);
```

SecureTransport Client Side

- Declaration of application-provided functions which perform actual network I/O—These get registered with SecureTransport as callbacks

```
extern OSStatus myReadCallback(
    SSLConnectionRef connection,
    void *data,
    size_t *dataLength);
extern OSStatus myWriteCallback(
    SSLConnectionRef connection,
    const void *data,
    size_t *dataLength);
```

SecureTransport Client Side

- Given a connected socket and a “GET” string, perform one secure transaction

```
OSStatus performSSL(
        int fd,                         // connected socket
        const char *sendData,           // to send to server
        size_t sendDataLen,
        char *rcvData,                  // received from server
        size_t rcvDataLen)              // size of *rcvData
{
        SSLContextRef sslCtx;
        OSStatus ortn;
        size_t actLen;
        size_t rcvBufRemaining = rcvDataLen;
```

SecureTransport Client Side

- Create an SSLContextRef and initialize it

```
SSLNewContext(false, &sslCtx);
SSLSetIOFuncs(sslCtx, myReadCallback,
    myWriteCallback);
SSLSetConnection(ctx, (SSLCOnnectionRef)sock);
```

SecureTransport Client Side

```
/* Options would be specified here, e.g.: */
SSLSetProtocolVersion(sslCtx, kSSLProtocol3Only);

/* Perform the SSL handshake */
do {
      ortn = SSLHandshake(ctx);
} while (ortn == errSSLWouldBlock);

/* Either handshake is complete, or an error
 * occurred */
if(ortn != noErr) {
/* error handling here */
}
```

SecureTransport Client Side

```
/* Send GET msg to server */
SSLWrite(sslCtx, sendData, sendDataLen, &actLen);

/* Fetch data from server */
while(rcvBufRemaining) {
        ortn = SSLRead(sslCtx,
                                rcvData,
        rcvBufRemaining,
        &actLen);
        if(ortn == errSSLWouldBlock) {
                /* in this loop, these are identical */
        ortn = noErr;
        }
        if(ortn != noErr) {
                /* Oops, error, abort */
                break;
        }
        /* process/save data and get some more */
        rcvBufRemaining -= actLen;
        rcvData += actLen;
}       /* filling rcvData */
```

SecureTransport Client Side

- Finish Up

```
/* finished, cleanup */
SSLClose(sslCtx);
SSLDisposeContext(sslCtx);
}
```

Who to Contact

Craig Keithley
Security and Cryptography Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2002/urls.html

Roadmap

110 Security: Authorization in Mac OS X: Using Authorization Services on Mac OS X	Civic **Wed., 2:00pm**
114 Security: Certificates in Mac OS X: Using X.509 certificates on Mac OS X	Civic **Thurs., 10:30am**
805 Introducing CFNetwork: Communicating with web services	Room C **Tue., 5:00pm**
814 Kerberos in Mac OS X: Learn about Kerberos on Mac OS X	Room C **Thurs., 5:00pm**
FF006 Security: Give us your feedback on security issues	Room J1 **Thurs., 2:00pm**

For More Information

- Apple Developer Security page
  http://developer.apple.com/macos/security.html
- Common Data Security Architecture
  http://opensource.apple.com
  http://www.opengroup.org
- PC/SC Documentation
  http://www.pcscworkgroup.com
- Product Security Web page
  http://support.apple.com/security
- Secure Trusted OS Consortium
  http://www.stosdarwin.org

Craig Keithley
Security and Cryptography Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002